Scalability Experiment

Recall the MK algorithm cannot work in online mode. Nevertheless, it may be instructive to consider the following. Suppose we could afford the luxury of storing all ten hours of data. Then we have a choice of invoking the MK algorithm, or running our algorithm. Thus we can ask:

- How *fast* we can do compared to the batch version of the same problem?

In order to address the scalability issue, we do the following experiment. Assume that we have 16 instances of the sparrow calls. Keeping this target instance count constant, we vary the number of points in our audio trace dataset from 0.01 million to 2.4 millions. As in the previous experiment, for each of these different size datasets we take a cache which can buffer at most 1/4000 subsequences possible. We run our sticky cache algorithm and the MK algorithm until we run out of data in all the datasets under question, and record the running time. We show the result below:

Running time comparison of sticky cache algorithm vs. MK algorithm. For smaller datasets, MK algorithm performs better. However, as soon as the dataset becomes large enough, our algorithm outperforms MK algorithm. The values for dataset sizes beyond 1.2 millions are extrapolated for the MK algorithm.

The figure above shows that the MK algorithm performs significantly better than the sticky cache algorithm when the dataset size is small. However, for dataset sizes beyond 0.75 million points, our approach starts performing better. The reason is straightforward, the MK algorithm is designed to detect *exact* motif pairs, in contrast to which it suffices detecting an approximate motif pair for the sticky cache algorithm. In addition to this, the code version of the MK algorithm we used is an optimized C code, whereas our implementation was in MATLAB. Even if this fact, we performed significantly better for very large datasets, which is a *harder* case for the MK algorithm.